DL-HW #06

2015004693_양상헌

실행환경: MAC OS TERMIINAL(MACBOOK PRO 2015 RETINA , MOJAVE 10.14.6) , ANACONDA

1. Source Code:

- *Assignment6 폴더 참조 (DL_HW_06week.py)*

```
#-> GAUSSIAN Noise
#X_noise = tf.Variable(tf.random_normal([n_input], mean = 0.0, stddev = 1.0)) #01
#X_noise = tf.Variable(tf.random_normal([n_input], mean = 0.0, stddev = 0.8)) #02
#X_noise = tf.Variable(tf.random_normal([n_input], mean = 0.0, stddev = 0.5)) #03
X_noise = tf.Variable(tf.random_normal([n_input], mean = 0.0, stddev = 0.2)) #04

#-> Uniform Noise
#X_noise = tf.Variable(tf.random_uniform([n_input], minval = -0.1, maxval = 0.1)) #05
#X_noise = tf.Variable(tf.random_uniform([n_input], minval = -0.05, maxval = 0.05)) #06
#X_noise = tf.Variable(tf.random_uniform([n_input], minval = -0.01, maxval = 0.01)) #07
#X_noise = tf.Variable(tf.random_uniform([n_input], minval = -0.005, maxval = 0.005)) #08

#add noise
X_noise_added = tf.add(X, X_noise)
```

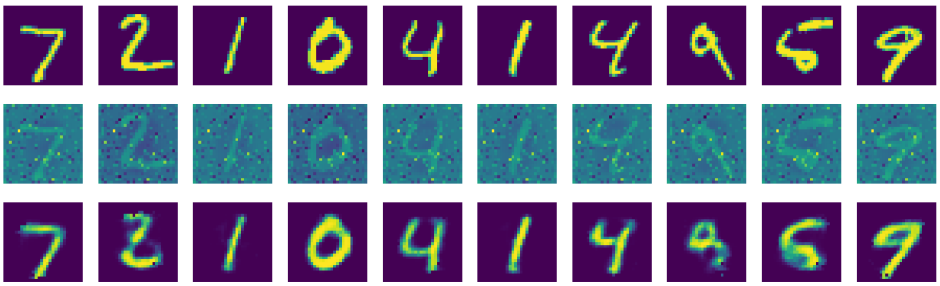
==== 중략 ====

```
with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    total_batch = int(mnist.train.num_examples/batch_size)

    for epoch in range(epoch_num):
        avg_cost = 0
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            _, cost_val = sess.run([optimizer, cost], feed_dict = {X: batch_xs})
            avg_cost += cost_val / total_batch
        print('Epoch:', '%d' % (epoch+1), 'cost:', '{:.9f}'.format(avg_cost))

    with_noise , samples = sess.run([X_noise_added , output], feed_dict = {X:
        mnist.test.images[:10]})

    fig, ax = plt.subplots(3, 10, figsize = (10,3))

    for i in range(10):
        ax[0][i].set_axis_off()
        ax[1][i].set_axis_off()
        ax[2][i].set_axis_off()
        ax[0][i].imshow(np.reshape(mnist.test.images[i], (28, 28)))
        ax[1][i].imshow(np.reshape(with_noise[i], (28, 28)))
        ax[2][i].imshow(np.reshape(samples[i], (28, 28)))
    plt.show()
```

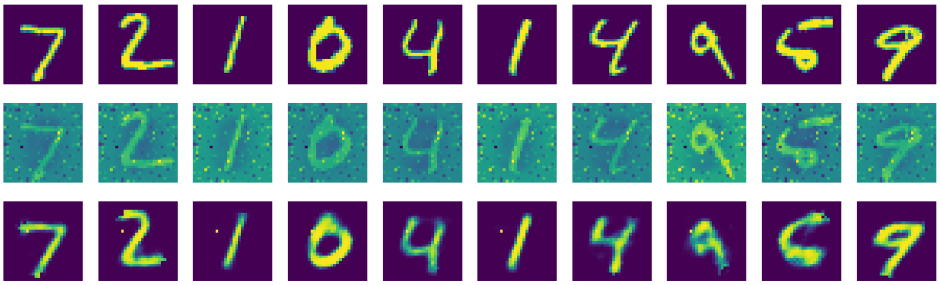
2. Result & Comparison:

<Normal Distribution Noise>

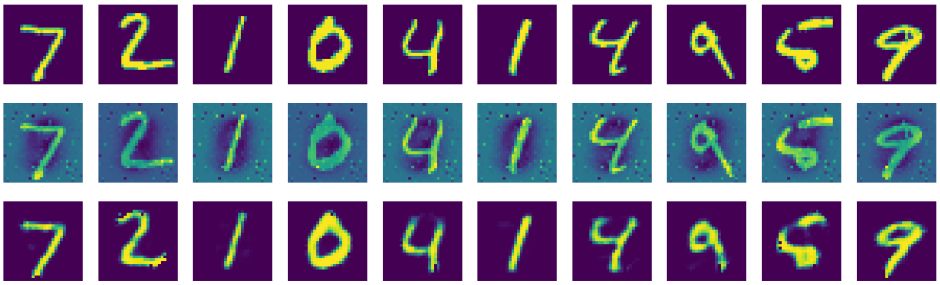
#01. Noise of Normal Distribution N(0, 1)



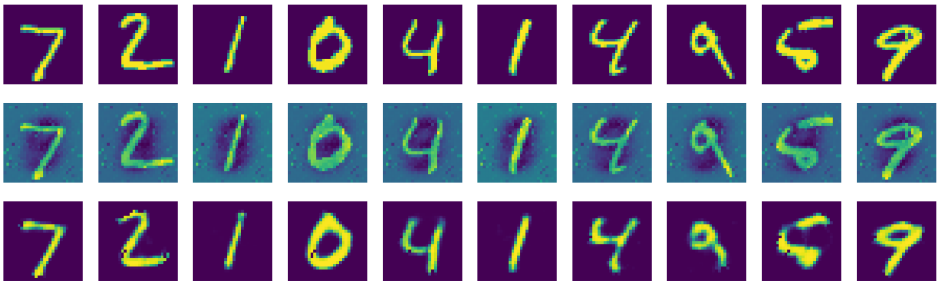
#02. Noise of Normal Distribution N(0, 0.8)



#03. Noise of Normal Distribution N(0, 0.5)

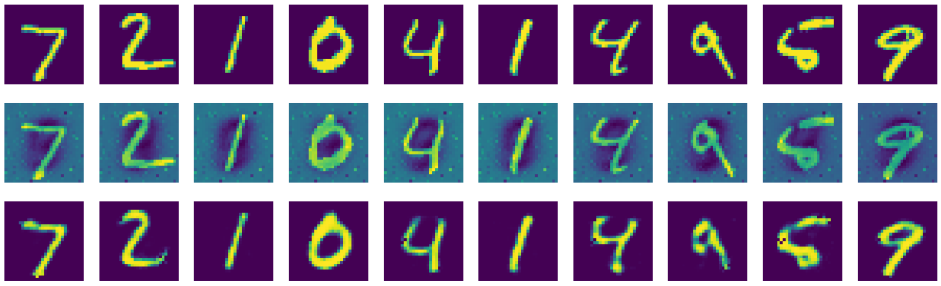


#04. Noise of Normal Distribution N(0, 0.2)

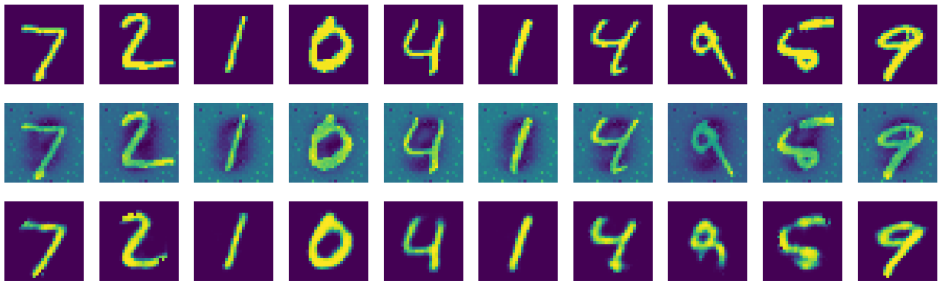


<Uniform Distribution Noise>

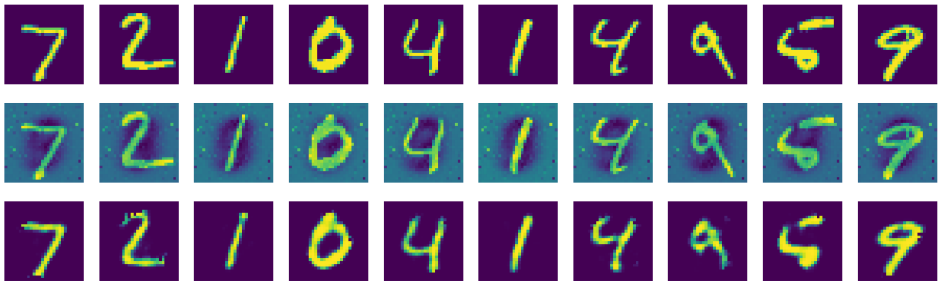
#05. Noise of Uniform Distribution in Range [-0.1, 0.1)



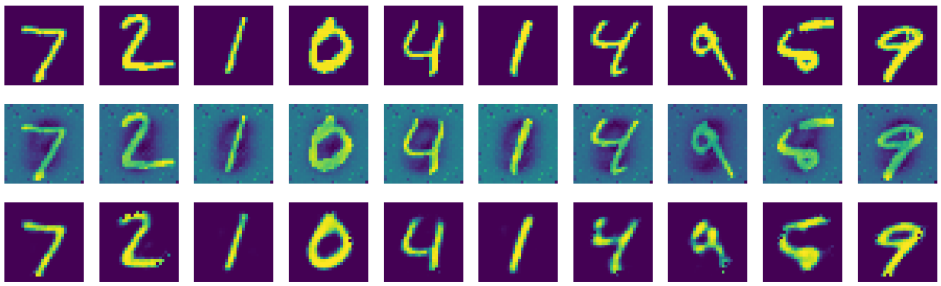
#06. Noise of Uniform Distribution in Range [-0.05, 0.05)



#07. Noise of Uniform Distribution in Range [-0.01, 0.01)



#08. Noise of Uniform Distribution in Range [-0.005, 0.005)



3. Disscussion:

< 코드 설명 >

전체 코드의 기본 구조는 수업자료 '5.AutoEncoder.pdf'에 나와있는 Stacked Auto Encoder 모델과 같은 구조를 사용 하였다. 차이점은 크게 인코딩이 진행 되기 직전에 Input data에 노이즈를 추가해주는 부분과, 학습이 끝난 후 10개의 Mnist data 로 test를 진핸한 결과를 사진으로 출력할 때 노이즈가 섞인 data의 사진도 같이 출력하도록 해준 부분이다.

1페이지 첫번째 사진은 인코딩이 진행되기 직전에 Input Data에 노이즈를 추가해주는 부분으로, 정규분포(Normal Distribution)를 따르는 노이즈와 균등분포(Uniform Distribution)를 따르는 노이즈를 사용하였다. 총 8가지의 노이즈를 사용하였는데 정규분포 노이즈 4종류와[N(0, 1), N(0, 0.8), N(0, 0.5), N(0, 0.2)], 균등분포 노이즈 4종류{ [-0.1, 0.1), [-0.05, 0.05), [-0.01, 0.01) [-0.005, 0.005) } 를 사용하였고 각각 노이즈에 따라 다른 결과를 비교하였다.

1페이지 두번째 사진은 학습 완료 후 테스트를 진행할 때 노이즈를 추가해준 데이터의 사진도 추가로 출력하도록 해준 부분이다.

< 결과 분석 >

결과 사진에서 첫번째 행의 이미지 10장은 TEST data로 사용되는 Mnist data로서 노이즈가 섞이지 않은 Input Data를 의미하고, 두번째 행의 이미지 10장은 첫번째 행의 이미지 data에 특정한 종류의 노이즈가 추가된 data의 이미지이다. (이때 첫번째 행의 각기 다른 Input Data 10개에 모두 같은 값의 노이즈가 추가된다.) 마지막으로 세번째 행의 이미지 10장은 두번째 행의 노이즈가 추가된 이미지 data들이 학습이 완료된 Stacked Auto Encoder-Decoder 모델을 거쳐 나오게 되는 결과값 이미지를 의미한다. (실제 현실에서 이상적인 노이즈는 정규분포를 따르는 노이즈라고 생각한다.)

#01~#04에서는 정규분포를 따르는 난수를 노이즈로 사용했고, **#01~#04**에서 각각의 노이즈 값의 평균값은 0으로 동일하지만 각기 다른 표준편차값(1, 0.8, 0.5, 0.2)을 사용하여 생성되는 노이즈 값에 차이를 주고 학습을 진행하였다.

노이즈의 분포에서 편차가 커질수록 인코딩-디코딩 시에 정확도가 낮아질 것이라 예상하고 학습을 진행 시켰고 테스트 결과값을 확인했을 때 예상과 비슷한 결과를 얻을 수 있었다. 2페이지 이미지들을 비교해 보면 (육안으로 확인해 보더라도) 표준편차값이 작아질 수록 더 정확한 결과를 내는 것을 확인 할 수 있다.

정규분포의 난수를 발생시킬 때 표준편차가 작아질 수록 평균값인 0에 근접한 난수가 발생할 확률이 높아지기 때문에, 표준편차가 작을 수록 노이즈를 추가한 이미지 데이타와 노이즈가 추가되지 않은 원래의 인풋 이미지 데이타 간의 차이가 줄어들게 되어서 위와 같은 결과를 낸다고 볼 수 있다.

#05~#08에서는 균등분포를 따르는 난수를 노이즈로 사용했고, 각기 다른 범위 { [-0.1, 0.1), [-0.05, 0.05), [-0.01, 0.01) [-0.005, 0.005) } 안에서 노이즈를 발생 시켰다.

노이즈의 범위가 넓을수록 정확도가 낮아질 것이라 예상하고 학습을 진행시켰는데 예상과는 조금 다르게 노이즈 발생 범위가 다른 4가지 경우 모두 어느정도 비슷한 정확도를 보였다. 3페이지 이미지들을 비교해보면 (육안으로 확인했을 때) 미미한 차이가 있을 뿐 큰 차이를 보이지 않았다.